INFORMATION AND COMPUTATION 80, 193–204 (1989)

The Iterated Mod Problem*

HOWARD J. KARLOFF

Department of Computer Science, University of Chicago, Chicago, Illinois 60637

AND

WALTER L. RUZZO

Department of Computer Science, University of Washington, Seattle, Washington 98195

The *iterated mod problem* is this: given a, $b_1, b_2, ..., b_n$, all integers or all polynomials in $\mathbb{Q}[x]$, is $((\cdots((a \bmod b_1) \bmod b_2)\cdots) \bmod b_n) = 0$? When the inputs are integers, we prove the problem *P*-complete with respect to log-space reductions, whereas in the polynomial case, we prove the problem is in *NC*. The significance of these results lies primarily in the similarity between the iterated mod problem and the Euclidean algorithm. We also show that the *superincreasing knapsack problem* is *P*-complete, using a very similar proof.

1. INTRODUCTION

We are interested in identifying the intrinsic complexity of natural problems with respect to general models of parallel computation. In this setting, the complexity class *NC*, a subset of the well-known class *P*, has emerged as an object of much theoretical interest. *NC* is the class of problems solvable "extremely rapidly," in time $\log^{O(1)} n$, on parallel machines with a "feasible," $n^{O(1)}$, number of processors. The class is reasonably model-independent, and contains a variety of interesting problems; see, e.g., (Cook, 1985). In this paper we investigate the parallel complexity of the *iterated mod problem*, to be defined shortly, which is related to and motivated by the ancient problem of computing the greatest common divisor (gcd) of two numbers. The parallel complexity of the gcd problem is poorly understood.

Most sequential algorithms for computing greatest common divisors are based on the Euclidean algorithm, the most common version of which is described as follows. Let $r_0 = a$, $r_1 = b$. Set

$$r_2 = r_0 \bmod r_1, \qquad r_3 = r_1 \bmod r_2, \qquad r_4 = r_2 \bmod r_3, \ldots.$$

* Research supported by NSF Grants ECS-8306622 and CCR-8703196.

0890-5401/89 $3.00

Then $\gcd(a, b)$ is the last nonzero r_i. This procedure works whether a and b are integers or univariate polynomials over $\mathbb{Q}$. More generally, it works in any Euclidean ring, assuming *mod* is appropriately defined.[1] In the integer case, the ratios $\lfloor r_i/r_{i+1} \rfloor$ are themselves important, since from them one can compute (in *NC*, in fact) the continued fraction expansion of a/b and integers α and β such that $\alpha a + \beta b = \gcd(a, b)$ (LeVeque, 1977).

(A. Borodin, J. von zur Gathen, and J. Hopcroft, 1982) proves that finding gcds of polynomials over arbitrary fields is in *Arithmetic-NC* (defined in Section 3), and in *NC* for finite fields and $\mathbb{Q}$. Yet, although numerous attempts have been made, no one has come close to finding an *NC* integer gcd algorithm, even a randomized one. The fastest parallel algorithm is (Chor and Goldreich, 1985), whose algorithm takes time $O(n/\log n)$ on n-bit numbers, while exploiting the power of concurrent access on a parallel RAM. The best parallel algorithm on a bounded-fan-in model is the $O(n)$ time systolic algorithm of (Brent and Kung, 1983). Thus, the integer gcd problem seems much harder than the polynomial one. Analogous situations exist for the sequential factoring and modular powering problems. Factoring integers is thought to be hard, but factoring rational polynomials is in *P* (Lenstra, Lenstra, and Lovász, 1982). Modular powering, i.e., computing $a^e \bmod b$ for n-bit integers e, is in *NC* for degree-n polynomials a and b over a finite field (Fich and Tompa, 1985), but is not known to be in *NC* for n-bit integers a and b, except in special cases (von zur Gathen, 1984). Like gcd, this problem is not known to be *P*-complete, nor is there other strong evidence that it is not in *NC*.

In this paper, we consider a problem related to the gcd problem, the *iterated mod problem*. We define it as follows: given a, b_1, b_2, ..., b_n, all positive integers or all in $\mathbb{F}[x]$, for some field $\mathbb{F}$, is

$$((\cdots((a \bmod b_1) \bmod b_2)\cdots) \bmod b_n) = 0?$$

(In the integer case, $s \bmod t$ is the least nonnegative integer congruent mod t to s.) Like the gcd problem, this one can be solved by setting

$$a_1 = a \bmod b_1, \quad a_2 = a_1 \bmod b_2, \quad a_3 = a_2 \bmod b_3, \ldots, \quad a_n = a_{n-1} \bmod b_n,$$

and then asking if $a_n = 0$. In some ways, this problem seems easier than the Euclidean algorithm, in that here the moduli $b_1, \ldots, b_n$ and the length of the computation are prespecified. On the other hand, the problem may be computationally harder, since the moduli are completely independent of each other, which is clearly not the case in the Euclidean algorithm.

[1] To be precise, all that is required of $r = a \bmod b$ is that $a = tb + r$ for some t, and either r is zero, or it is smaller than b under the norm associated with the ring. See, e.g., (Herstein, 1964).

In Section 3, we prove that the polynomial iterated mod problem is in *NC*. In contrast, in Section 2 we show that an *NC* algorithm is very unlikely to exist for the integer version, thereby strengthening the apparent dichotomy between integer and polynomial problems.

To be more specific, in Section 2 we show that the integer iterated mod problem is complete for *P* with respect to log-space reductions. The *P*-complete problems form another interesting subclass of *P*, whose members are believed to be in some sense "inherently sequential." In particular, if any *P*-complete problem were shown to be in *NC*, then $P = NC$, a possibility most researchers consider unlikely. Thus, an *NC* algorithm exists for integer iterated mod if and only if $P = NC$. *P*-completeness has similar implications for (sequential) space complexity: a *P*-complete problem such as integer iterated mod can be in $DSPACE((\log n)^c)$ if and only if $P \subset DSPACE((\log n)^c)$, also considered unlikely.

The result also shows something about the complexity of the mod function. Analogous problems not involving mod do not become *P*-complete until they are made much more general. For example, the integer iterated mod problem can be viewed as a formula whose only connectives are integer mod operations. From the *P*-completeness result, we conclude that an efficient parallel algorithm to evaluate such formulae probably does not exist. In contrast, we can efficiently parallelize evaluation of Boolean formulae, or of integer formulae with the seemingly richer set of operations $\{+, -, \times\}$, or even of Boolean or integer straight-line programs under suitable degree constraints (see Valiant, Skyum, Berkowitz, and Rackoff, 1983). These straight-line program value problems finally become *P*-hard when the degree constraint is lifted. This was shown in (Ladner, 1975) in the Boolean case, and is easily seen in the arithmetic case by reduction from the *NAND*-gate circuit value problem discussed below, and the observation that $1 - x \cdot y$ computes the *NAND* of 0, 1-valued variables x and y (Venkateswaran, 1983).

In Section 4 we show, using related proof techniques, that the *superincreasing knapsack problem* is also *P*-complete.

See (Garey and Johnson, 1979), or (Cook, 1985) for more discussion of the significance of *P*-completeness results.

2. The Integer Iterated Mod Problem is *P*-Complete

To simplify our reduction, we alter our notation. We assume we are given positive integers $x, m_n, m_{n-1}, \ldots, m_1$, and we wish to determine if

$$((\cdots((x \bmod m_n) \bmod m_{n-1})\cdots) \bmod m_1) = 0.$$

Clearly this problem is in *P*.

Our reduction is from the *NAND*-gate circuit value problem. An instance consists of a directed acyclic graph, having r nodes $y_1, ..., y_r$ of indegree 0 (the *inputs*), and G nodes, numbered $G, G-1, ..., 1$, of indegree 2 known as *gates*. We assume the gates are numbered in reverse topological order, i.e., every edge is directed from an input to a gate, or from a higher numbered gate to a lower numbered one. The last gate, the gate numbered 1, is known as the *output* and from it we imagine a single artificial outgoing edge, one whose head is unspecified. The $E = 2G + 1$ edges are numbered so that the two edges into gate g have labels $2g$ and $2g-1$, and the artificial edge out of the output gate is labeled 0. Furthermore, we are given a value $Y_l \in \{0, 1\}$ for each input y_l. Each gate, computing the *NAND* of the two bits fed to it, acquires a binary value in the obvious way, and each edge "carries" a binary value. Our goal is to compute the value of the output gate. That the problem is *P*-complete can be seen by modifying the proof in (Ladner, 1975).

The reduction from the circuit-value problem to the integer iterated mod problem is as follows. Let $n = 2G$. Let x be the E-bit integer whose jth bit (with place value 2^j) is Y_l if edge j is incident from input y_l, otherwise, the jth bit of x is 1. For $1 \leqslant g \leqslant G$, let

$$m_{2g} = 2^{2g} + 2^{2g-1} + \sum_{\substack{\text{edge } j \text{ is an out-edge} \\ \text{"from" gate } g}} 2^j,$$

and

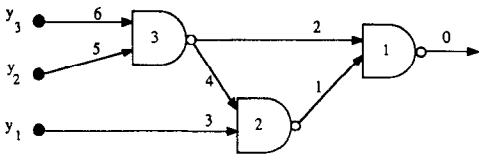
$$m_{2g-1} = 2^{2g-1}.$$

Certainly, this is a log-space reduction.

Intuitively, the reduction works as follows. A pair of moduli m_{2g} and m_{2g-1} simulate gate g. The topological numbering of gates and edges ensures that at the appropriate time, the two inputs to *NAND* gate g will be the two "high-order" bits of the residue of x. The two leading 1 bits in m_{2g} have the effect of testing these two inputs when reducing mod m_{2g}, which thereby simulates the *NAND*. Specifically, if the two "high-order" bits are both 1, then the low order bits of m_{2g} clear the bits corresponding to the out-edges from gate g. Reducing mod m_{2g-1} is a clean-up step that clears the two "high-order" bits if only one of them was a 1. We illustrate the construction, and the associated iterated mod computation, with the example in Fig. 1.

The next theorem establishes the correctness of the reduction, by proving that the output gate of the circuit-value instance has value 0 if and only if

$$((\cdots((x \bmod m_n) \bmod m_{n-1})\cdots) \bmod m_1) = 0.$$



The Construction								Simulation on Input $Y_1 = Y_2 = Y_3 = 1$							
bit/edge	6	5	4	3	2	1	0	bit/edge	6	5	4	3	2	1	0
x	Y_3	Y_2	1	Y_1	1	1	1	$x \cdots$	1	1	1	1	1	1	1
m_6	1	1	1	0	1	0	0	$\cdots \bmod m_6$	0	0	0	1	0	1	1
m_5	0	1	0	0	0	0	0	$\cdots \bmod m_5$			0	1	0	1	1
m_4			1	1	0	1	0	$\cdots \bmod m_4$			0	1	0	1	1
m_3			0	1	0	0	0	$\cdots \bmod m_3$					0	1	1
m_2					1	1	1	$\cdots \bmod m_2$					0	1	1
m_1					0	1	0	$\cdots \bmod m_1$							1

FIG. 1. Example of the reduction.

THEOREM. *Let* $x_g = ((\cdots((x \bmod m_n) \bmod m_{n-1})\cdots \bmod m_{2g})$ $\bmod m_{2g-1})$, *for* $1 \leqslant g \leqslant G$, *and let* $x_{G+1} = x$.

(a) *For each* g, $1 \leqslant g \leqslant G+1$, $x_g < 2^{2g-1}$.

(b) *Let* $1 \leqslant g \leqslant G+1$, $0 \leqslant j \leqslant 2g-2$. *If edge* j *is an outgoing edge from an input node, or from a gate* h *with* $h \geqslant g$, *then* x_g's jth *bit is the value carried by edge* j. *If not,* x_g's jth *bit is* 1.

Proof. By downward induction on g. The basis case, $g = G+1$, is easy. For the induction step, let $1 \leqslant k \leqslant G$ and suppose that the theorem holds for all $g > k$. We prove it also holds when $g = k$. Note that $x_k = (x_{k+1} \bmod m_{2k}) \bmod m_{2k-1}$. Since $m_{2k-1} = 2^{2k-1}$, x_k consists of the $2k-1$ least-significant bits of $(x_{k+1} \bmod m_{2k})$, and part (a) follows trivially.

We need to show (b) when $k = g$ for all j in the range $0 \leqslant j \leqslant 2k-2$. Note that by the induction hypothesis, the only bits which may differ between x_{k+1} and x_k are those bits corresponding to (in- or out-) edges incident to vertex k. Edges $2k$ and $2k-1$ are the in-edges to gate k. By part (b) with $g = k+1$, we infer that x_{k+1}'s $(2k)$th bit is the value carried by edge $2k$, and that x_{k+1}'s $(2k-1)$st bit is the value carried by edge $2k-1$. These bits will be zero in x_k, by part (a). Part (b) applied with $g = k+1$ also implies that if edge j is an out-edge of gate k, then x_{k+1}'s jth bit is a 1, whereas these bits in x_k must be the value of gate k. All other bits must be unchanged. There are two cases to consider, depending on bits $2k$ and $2k-1$ (the "high-order" two bits) of x_{k+1}.

Case 1. In x_{k+1}, bits $2k$ and $2k-1$ are 1. Consequently, $x_{k+1} \geqslant m_{2k}$, yet $x_{k+1} < 2m_{2k}$ and therefore $(x_{k+1} \bmod m_{2k}) = x_{k+1} - m_{2k}$. Since taking mod $m_{2k-1} = 2^{2k-1}$ does nothing, we conclude that x_k is obtained from x_{k+1} by deleting the leading two 1's, and replacing the 1 in position j by a 0, for every out-edge j leaving gate k; all other bits are unchanged. But since the two edges into gate k were carrying 1's, and $NAND(1, 1) = 0$, (b) now holds for $g = k$.

Case 2. In x_{k+1}, at least one of bits $2k$ and $2k-1$ is a 0. Then $x_{k+1} < m_{2k}$ (because m_{2k} has 1's in positions $2k$ and $2k-1$), so $(x_{k+1} \bmod m_{2k}) = x_{k+1}$. Hence x_k consists of the rightmost $2k-1$ bits of x_{k+1}, and in position j of x_k, where j is an out-edge of gate k, we still have a 1. Because $NAND(1, 0) = NAND(0, 1) = NAND(0, 0) = 1$, (b) holds for $g = k$.

This establishes condition (b), and completes the proof. ∎

It might seem at first that the completeness of the integer iterated mod problem hinges on the ability to specify n different moduli. In fact, this is not entirely true. The problem remains complete if, for some constant k, the modulus sequence consists of n/k "copies" of a sequence of k distinct moduli, where each modulus in each copy is a shifted version of one of the original k moduli. Equivalently, there is a constant k, independent of n, and k integers $m_0(n), \ldots, m_{k-1}(n)$ depending on n and computable from n in $\log n$ space, such that the following problem involving only k distinct moduli is P-complete: Given an n-bit integer x, compute

for $i := 1$ **to** n **do** $x := 2x \bmod m_{i \bmod k}(n)$; **if** $x = 0$ **then accept.**

The proofs of these results are simple modifications of the construction above, based on the observation that the circuits produced from P-time Turing machines in the proof in (Ladner, 1975) that the circuit value problem is complete are in fact extremely regular in structure. We omit the details.

The P-completeness of integer iterated mod *does* appear to depend on the presence of n-bit moduli, since the problem restricted to moduli of length $O(\log n)$ bits, or even $\log^{O(1)} n$ bits, is in NC. In other words, the problem is not "strongly" P-complete unless $P = NC$. Basically, the restricted problem is in NC since the magnitude of the residue is at least halved by every consecutive pair of "non-trivial" mod operations, and the next "non-trivial" modulus may be found by binary search.

3. The Polynomial Iterated Mod Problem Is in *NC*

The *polynomial iterated mod problem* is: given univariate polynomials $a(x), b_1(x), ..., b_n(x)$ over a field $\mathbb{F}$, compute the residue

$$((\cdots((a(x) \bmod b_1(x)) \bmod b_2(x)\cdots) \bmod b_n(x)).$$

Does the *P*-completeness proof from the previous section apply to this problem also? No. Basically, since the degree of a polynomial is determined by only one "high-order bit," a polynomial mod cannot test two "bits." For example, for integers we had $(10)_2 \bmod (11)_2 = (10)_2$, but in the analogous case for polynomials, we have $(1\cdot x^2 + 0\cdot x) \bmod (1\cdot x^2 + 1\cdot x) = (0\cdot x^2 - 1\cdot x)$. Not only does our *P*-completeness proof fail in the polynomial case, but in this section, we will show that there is a very fast parallel algorithm for this problem.

For an algebraic problem of this type, it is natural to assume a model of computation in which inputs and outputs are elements of the field $\mathbb{F}$, rather than bits, and in which each processor may compute the primitive field operations in unit time. The analog of *NC* for such models usually is called *Arithmetic–NC*, and *Arithmetic–NC*^{k} denotes problems solvable by such models in time $\log^k n$ with $n^{O(1)}$ processors. To be more precise, "time" here means depth of a uniform family of polynomial-sized arithmetic circuits, i.e., circuits having gates which compute primitive field operations. It is often possible to translate *Arithmetic–NC* algorithms for arbitrary fields into *NC* algorithms for some particular fields, including finite fields and $\mathbb{Q}$, whose elements are easily encoded and manipulated as bit strings. This is true, for instance, of the algorithm for solving linear systems given in (Borodin, von zur Gathen, and Hopcroft, 1982), and of their polynomial gcd algorithm. Note, however, that for *NC* computations over $\mathbb{Q}$ we generally cannot insist on answers in lowest terms, unless and until integer gcd is shown to be in *NC*.

In this section we will show that the polynomial iterated mod problem over any field $\mathbb{F}$ is in *Arithmetic–NC*^{2}. In the important cases where $\mathbb{F}$ is a (fixed) finite field, or the rationals $\mathbb{Q}$, the problem will be shown to be in *NC*^{2}, i.e., solvable by *Boolean* circuits of polynomial size and log-squared depth. The proof in both cases is by reduction to the problem of solving non-singular triangular systems of linear equations over $\mathbb{F}$, a problem which is known to be in *Arithmetic–NC*^{2} in general, and in *NC*^{2} for some particular fields, including all finite fields and $\mathbb{Q}$. (The original *Arithmetic–NC*^{2} algorithm is found in (Heller, 1974); see (Heller, 1978) for a survey of results on this problem, and (Greenberg, Ladner, Paterson, and Galil, 1982) for a more processor-efficient solution. *NC*^{2} algorithms for particular rings are presented by (Borodin, von zur Gathen, and Hopcroft, 1982), and

by (Borodin, Cook, and Pippenger, 1983), and in fact work for general linear systems.)

Note that we have *not* phrased the polynomial iterated mod problem as a decision problem, such as deciding whether the residue is identically zero. Intuitively, the problem of computing the residue is at least as hard as this decision problem, so our result is intuitively stronger when phrased this way. In fact, the associated decision problem is *not* in *Arithmetic–NC*, but only for the trivial technical reason that the *Arithmetic–NC* model does not allow tests.

Let $b_0(x) = r_0(x) = a(x)$, and for $0 \leqslant i \leqslant n$ let $d_i = \text{degree}(b_i)$. Without loss of generality, assume $d_0 \geqslant d_1 > d_2 > \cdots > d_n$. Now, there exist unique polynomials $q_i(x)$ and $r_i(x)$ for $1 \leqslant i \leqslant n$ such that

$$r_{i-1}(x) = q_i(x) \cdot b_i(x) + r_i(x), \quad \text{with } \text{degree}(r_i) < \text{degree}(b_i) = d_i \text{ (or } r_i = 0). \tag{3.1}$$

Namely, q_i and r_i are the quotient and remainder, respectively, of r_{i-1} divided by b_i. Further,

$$a(x) = q_1(x) \cdot b_1(x) + q_2(x) \cdot b_2(x) + \cdots + q_n(x) \cdot b_n(x) + r_n(x), \tag{3.2}$$

and

$$r_n(x) = ((\cdots((a(x) \bmod b_1(x)) \bmod b_2(x)) \cdots) \bmod b_n(x)). \tag{3.3}$$

Let a_j, $(b_{i,j}, q_{i,j}, r_{i,j})$ be the coefficient of x^j in $a(x)$, $(b_i(x), q_i(x), r_i(x)$, respectively). Then, we can find the coefficients of r_n (and simultaneously of all the q_i's) by solving the $(d_0 + 1) \times (d_0 + 1)$ triangular linear system shown in Fig. 2.

This linear system arises from Eq. (3.2) above by simply equating the coefficients of like powers of x. For example, the term $q_1(x) \cdot b_1(x)$ in the right-hand side of (3.2) gives rise to the leftmost block of $(d_0 - d_1 + 1)$ columns in the matrix, containing the coefficients of $b_1(x)$, and the top block of a like number of entries in the right hand column vector, containing the coefficients of $q_1(x)$.

Notice that the degree of $q_1(x)$ is exactly $d_0 - d_1$, so we have included the contributions of all non-zero $q_{1,j}$'s. Similarly, the ith block of columns, $2 \leqslant i \leqslant n$, corresponds to the term $q_i(x) \cdot b_i(x)$ in Eq. (3.2), and the last block to $r_n(x)$. Again, from Eq. (3.1) the degree of r_{i-1} is strictly less than d_{i-1}, so the degree of $q_i(x)$ can be at most $d_{i-1} - d_i - 1$. Thus we have included all non-zero $q_{i,j}$'s. A similar argument holds for the coefficients of $r_n(x)$.

That this system of equations has a unique solution follows from the uniqueness of the solution to Eq. (3.1). It can also be seen directly, since it is a

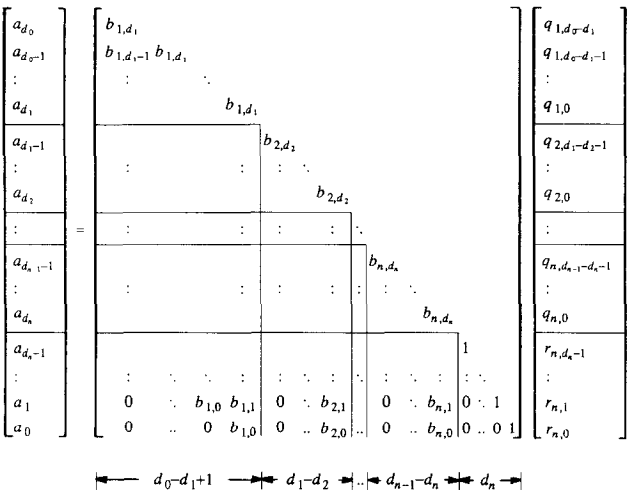


FIG. 2. Solving the polynomial iterated mod problem.

triangular matrix whose diagonal entries are all non-zero, and hence has a non-zero determinant. The diagonal entries b_{i,d_i} are nonzero, since $b_i(x)$ has degree d_i by definition.

We remark that, if desired, it is easy in parallel to recover the rest of the r_i's, once the q_i's are known. It is also worth remarking that if all the given $b_i(x)$ are monic, i.e., $b_{i,d_i} = 1$, then the resulting matrix is invertible over *any* commutative ring, not just a field, and further the inverse computation is still in *Arithmetic–NC*.

4. THE SUPERINCREASING KNAPSACK PROBLEM IS *P*-COMPLETE

The well-known 0–1 *knapsack problem* is: given an integer w, and a sequence of integers $w_1, w_2, ..., w_n$, is there a sequence of 0–1 valued variables $x_1, x_2, ..., x_n$ such that $w = \sum_{j=1}^{n} x_j \cdot w_j$. This problem is known to be *NP*-complete (Garey and Johnson, 1979). A knapsack instance is called *superincreasing* if each weight w_i is larger than the sum of the previous weights: for all $2 \leqslant i \leqslant n$ we have $w_i > \sum_{j=1}^{i-1} w_j$. A simple "greedy" strategy, considering the w_i's in decreasing order, gives a linear time algorithm for solving superincreasing knapsacks. Superincreasing knapsacks are best known as the basis for the Merkle–Hellman cryptosystem (Merkle and Hellman, 1978) (subsequently shown to be of questionable security (Shamir, 1982; Adleman, 1983)).

Larry Carter and Victor Miller (personal communication) pointed out to us that our *P*-completeness proof for the iterated mod problem is easily

modified to show that the superincreasing knapsack problem is also *P*-complete. We will sketch this proof below.

The reduction is again from the *NAND*-gate circuit value problem. We modify the construction given earlier in two ways. First, we use "radix 4"; second, we add a third modulus for each gate g. Its value will lie between that of m_{2g} and m_{2g-1}, so we will call it $m_{2g-0.5}$. Specifically, we let

$$m_{2g} = 4^{2g} + 4^{2g-1} + \sum_{\substack{\text{edge } j \text{ is an out-edge} \\ \text{"from" gate } g}} 4^j,$$

$$m_{2g-0.5} = 4^{2g} - 4^{2g-1},$$

and

$$m_{2g-1} = \quad 4^{2g-1}.$$

Since $m_{2g-0.5}$ is less than m_{2g} and divisible by m_{2g-1}, it is not hard to see that the construction is still correct. One effect of this change is that now all *quotients* corresponding to the remainder sequence computed by iterated mod are either 0 or 1. (They were previously 0, 1, or 2.) Further, it is not hard to see that the modified modulus sequence (taken in reverse) is superincreasing: the "high order" radix-4 digits of consecutive moduli are $(01)_4$, $(03)_4$, $(11)_4$, and $(0100)_4$, ..., which are superincreasing, regardless of the lower order digits of $(0011)_4$, or of the lower order digits contributed by earlier moduli. The resulting iterated mod problem has value zero if and only if the corresponding superincreasing 0–1 knapsack problem has a solution. This completes the proof sketch.

Our proof can also be modified to show the *P*-completeness of the *first fit decreasing* heuristic for the *bin packing problem.* Using similar techniques, this was previously shown *P*-complete in (Warmuth, 1984). Independently, (Gopalakrishnan, 1986) showed similar results for the first-fit and best-fit heuristics for bin packing. The first fit decreasing problem is also known to be "strongly" *P*-complete, i.e., it remains *P*-complete when the object weights are given in unary rather than binary. The latter requires stronger proof techniques (Anderson, 1986; Anderson, Mayr, and Warmuth, 1988). Another result using a proof technique similar to ours is the *P*-completeness of the *list scheduling problem*, shown in (Helmbold and Mayr, 1984).

5. Conclusions

The main problem we have studied in this paper was motivated by a desire to understand the parallel complexity of the integer gcd problem. While the gcd problem remains open, we have provided strong evidence

that the related integer iterated mod problem is not highly parallelizable. To our knowledge, this is the first *P*-completeness result for such a highly structured algebraic problem over the integers. Hopefully, it will serve as a stepping stone to such results for other integer problems, perhaps including gcd, or powering.

RECEIVED September 29, 1986; ACCEPTED October 31, 1986

REFERENCES

ADLEMAN, L. M. (1983), On breaking generalized knapsack public key cryptosystems, *in* "Proceedings, Fifteenth Annu. ACM Symp. on Theory of Computing," pp. 402–412.

ANDERSON, R. (1986), "The Complexity of Parallel Algorithms," Ph.D. dissertation, Stanford University; Technical Report STAN-CS-86-1092.

ANDERSON, R. MAYR, E., AND WARMUTH, M. (1988), "Parallel approximation algorithms for bin packing," Department of Computer Science, Technical Report STAN-CS-88-1200, Stanford University; *Inform. and Comput.*, in press.

BORODIN, A., COOK, S., AND PIGGENGER, N. (1983), "Parallel Computation for Well-Endowed Rings and Space-Bounded Probabilistic Machines," Department of Computer Science Technical Report 162/83, University of Toronto.

BORODIN, A., VON ZUR GATHEN, J., AND HOPCROFT, J. (1982), Fast parallel matrix and gcd computations, *Inform. and Control* **52**, 241–256.

BRENT, R. P., AND KUNG, H. T. (1983), Systolic VLSI arrays for linear time gcd computation, *in* "VLSI 83, IFIP" (F. Anceau and E. J. Aas, Eds.), pp. 145–154, Elsevier Science, New York.

COOK, S. A. (1985), A taxonomy of problems with fast parallel algorithms, *Inform. and Control* **64**, Nos. 1–3, 2–22.

CHOR, B., AND GOLDREICH, O. (1985), An improved parallel algorithm for integer gcd, preprint, MIT Laboratory for Computer Science.

FICH, F. E., AND TOMPA, M. P. (1985), The parallel complexity of exponentiating polynomials over finite fields, *J. Assoc. Comput. Mach.* **35**, No. 3, 651–667.

GOPALAKRISHNAN, P. S. (1986), "Parallel Approximate Algorithms for Combinatorially Hard Problems," Ph.D. dissertation, Department of Computer Science, University of Maryland.

GAREY, M. R., AND JOHNSON, D. S. (1979), "Computers and Intractability: A Guide to the Theory of *NP*-Completeness," Freeman, San Francisco.

GREENBERG, A. C., LADNER, R. E., PATERSON, M. S., AND GALIL, Z. (1982), Efficient parallel algorithms for linear recurrence computation, *Inform. Process. Lett.* **15**, No. 1, 31–35.

HELLER, D. (1974), A determinant theorem with applications to parallel algorithms, *SIAM J. Numer. Anal.* **11**, 559–568.

HELLER, D. (1978), A survey of parallel algorithms in numerical linear algebra, *SIAM Rev.* **20**, No. 4, 740–777.

HERSTEIN, I. N. (1964), "Topics in Algebra," Blaisdell, New York, p. 110.

HELMBOLD, D., AND MAYR, E. (1984), "Fast Scheduling Algorithms on Parallel Computers," Department of Computer Science, Technical Report STAN-CS-84-1025, Stanford University.

LADNER, R. E. (1975), The circuit value problem is log space complete for *P*, *SIGACT News* **7**, 18–20.

LENSTRA, A. K., LENSTRA, H. W., JR., AND LOVÁSZ, L. (1982), Factoring polynomials with rational coefficients, *Math. Ann.* **261**, 515–534.

LEVEQUE, W. J. (1977), "Fundamentals of Number Theory," pp. 226–231, Addison–Wesley, Menlo Park, CA.

MERKLE, R. C., AND HELLMAN, M. E. (1978), Hiding information and signatures in trapdoor knapsacks, *IEEE Trans. Inform. Theory* **IT-24**, 525–530.

SHAMIR, A. (1982), A polynomial time algorithm for breaking the basic Merkle–Hellman cryptosystem, *in* "Proceedings, Twenty Third Annual IEEE Symp. on Foundations of Computer Science," pp. 145–152.

VENKATESWARAN, H. (1983), personal communication.

VALIANT, L. G., SKYUM, S., BERKOWITZ, S., AND RACKOFF, C. (1983), Fast parallel computation of polynomials using few processors, *SIAM J. Comput.* **12**, No. 4, 641–644.

VON ZUR GATHEN, J. (1984), Computing powers in parallel, *SIAM J. Comput.* **16**, No. 5, 930–945.

WARMUTH, M. (1984), Parallel approximation algorithms for one-dimensional bin packing, manuscript.